Turing-completeness of additive transformations in the ETS formalism[1]

Oleg Golubitsky

Department of Mathematics
University of Pisa
Via Buonarroti 2, 56127, Pisa, Italy
golub@dm.unipi.it

Abstract

We represent strings over a finite alphabet by the data structures, called string structs, introduced in the evolving transformation system (ETS) model, which was proposed as a general model for structural object representation and classification and has been formalized recently. Then we describe string languages as classes of structs generated by a finite set of context-sensitive transformations. The transformations are assumed to be additive, or non-deleting, in order to maintain the explicitness of representation of the constructive history of objects from a class, postulated in the ETS formalism. In case of strings, additive transformations are context-sensitive insertions; however, in case of structs they are more general. We prove that any recursive language can be represented as a class of string structs generated by additive transforms.

1 Introduction

The evolving transformation system (ETS) model is a model for structural object representation. It was proposed in 1990 by Lev Goldfarb [5, 4] to be applied to any pattern learning or classification problem. The model is based on the postulate that every real-world object is a result of a constructive process called the object's constructive (sometimes evolutionary or developmental) history. Moreover, it is postulated that the representation of this constructive process should be explicitly present in the object's representation.

Recently, the first formalization of the model appeared in several versions [7, 6, 8]. The version [8] considered in this paper defines two main data structures. First, *formations* are defined to represent constructive processes, i.e., finite sequences of elementary constructive steps that can generate objects from a certain broad class. Generally speaking, several different constructive process may produce the same object. Therefore, a *semantic equivalence relation* on formations is introduced, with respect to which two formations are equivalent if and only if the corresponding constructive processes produce the same object. Finally, *structs* are defined to represent objects as equivalence classes of formations w.r.t. the semantic equivalence relation. Thus, a struct is a data structure that represents a real-world object. It is shown in [8] that strings, trees, and graphs are particular cases of structs.

In order to describe classes of objects, context-sensitive transforms are defined on formations, and then carried over to structs. A class is defined as a set of all structs generated by a given finite set of transforms. As opposed to the approaches based on

[1]The work has been supported by the Progetto di Interesse Nazionale "Algebra Commutativa e Computazionale" of the Italian "Ministero dell'Istruzione dell'Università della Ricerca Scientifica Tecnologica and by the RFBR grant no. 02-01-01033.

the Chomsky formal grammars [3, 9], all transforms in the ETS formalism are assumed to be *additive*, or non-deleting. This restriction ensures that the transforms do not erase the constructive history of objects, whose explicit presence in object representation is required by the above postulate. As a consequence, the problem of inductive learning of transforms from a finite training set is also simplified (see, for example, [1] for a learning algorithm developed for classes of strings generated by additive transforms).

In case of conventional formalisms (strings, trees, and graphs), the restriction of additivity significantly reduces the generative power of transforms. However, as we show in this paper, in case of general structs, the restriction of additivity does not imply any restriction of generative power whatsoever, i.e., arbitrary recursive languages can be generated.

The structure of the paper is as follows. In Section 2, the concepts of recursive language and Turing machine are reviewed. In Section 3, the main concepts of the ETS formalism are introduced. The example carried through Section 3 shows how to simulate Turing machines by additive transforms in the ETS formalism. In Section 4, correctness of this construction is proved, which implies that any recursive language can be generated by additive transforms in the ETS formalism.

2 Recursive languages and Turing machines

A *Turing machine* M (see also [10]) is specified by

- A finite alphabet Σ containing a special "blank" symbol $\Box$.
- A finite set Q, whose elements are called "states", containing three special elements: the initial state q_0, the "yes"-state q_Y and the "no"-state q_N.
- A transition mapping $\delta : (Q \setminus \{q_Y, q_N\}) \times \Sigma \to Q \times \Sigma \times \{L, R\}$.

The machine consists of an infinite tape subdivided into cells containing characters from Σ and a head that points to a single cell at each moment. Altogether, the contents of the machine's tape, the position of the head, and the current state are called the *configuration* of the Turing machine.

The initial configuration is the following: the machine's tape contains a finite string $x \in (\Sigma \setminus \{\Box\})^*$, called the *input string*, surrounded by blank symbols, the head points to the leftmost character of x, and the state is q_0. The machine can make a *step* as follows. Given that, in the current configuration, the machine's state is $q \in Q \setminus \{q_Y, q_N\}$ and the head points to a cell containing character $c \in \Sigma$, perform the following operations:

- compute $\delta(c, q) = (c', q', d)$
- write character c' in the cell to which the head points
- move the head one position left, if $d = L$, or one position right, if $d = R$
- enter state q'.

If the machine enters state q_Y or q_N, it halts.

We will specify the transition mapping of a Turing machine by listing a set of commands of the form

$$(q, c) \mapsto (q', c', d),$$

where $q \in Q \setminus \{q_Y, q_N\}$, $c, c' \in \Sigma$, $q' \in Q$, $d \in \{L, R\}$. The presence of the above command in the list means that $\delta(q, c) = (q', c', d)$. For every pair $(q, c) \in (Q \setminus \{q_Y, q_N\}) \times \Sigma$, there is at most one command in the list, and if there are none, we assume that $\delta(q, c) = (q_N, \square, L)$.

The machine *accepts* a language $L \subset (\Sigma \setminus \{\square\})^*$, if

- For all $x \in L$, the machine, starting in the initial configuration with the input string x, halts in state q_Y after finitely many steps.
- For all $x \in (\Sigma \setminus \{\square\})^* \setminus L$, the machine, starting in the initial configuration with the input string x, halts in state q_N after finitely many steps.

A language is called *recursive*, if there exists a Turing machine that accepts it. Informally, a language L is recursive, if there exists an algorithm that decides for any input string whether it belongs to L or not.

Given a Turing machine M accepting a language L, we would like to modify it, so that

- for every configuration of M, the tape contains a string from $(\Sigma \setminus \{\square\})^*$ surrounded by blank symbols
- M erases everything from the tape before halting in state q_Y or q_N.

This can be achieved through the following modifications:

- add states $q_{Y_1}, q_{Y_2}, q_{N_1}, q_{N_2}, q'_Y, q'_N, q_{YE}, q_{NE}$ to Q
- add a new character X to Σ
- for every command of the form $(q, \square) \mapsto (q', c', d)$, add command $(q, X) \to (q', c', d)$
- replace every command of the form $(q, c) \mapsto (q', \square, d)$ by $(q, c) \mapsto (q', X, d)$.

These modifications ensure that for every configuration of M, the tape contains a string from $(\Sigma \setminus \{\square\})^*$, and the head points to a character in the string or to the blank character immediately preceding or following the string. Next, replace states q_Y by q'_Y and q_N by q'_N in all commands of M and add the following commands:

$$\begin{array}{ll} (q'_Y, \alpha) \mapsto (q_{Y_1}, \alpha, R) & \alpha \in \Sigma \\ (q_{Y_1}, \square) \mapsto (q_{Y_1}, \square, L) & \end{array}$$

(and similar commands for the "no"-state). This ensures that, when the machine enters q_{Y_1} or q_{N_1}, the head points to a character in the string. Finally, add the following commands (again, we list only the commands for the "yes"-state, and the corresponding commands for the "no"-state are assumed implicitly):

$$\begin{array}{ll} (q_{Y_1}, \alpha) \mapsto (q_{Y_2}, \alpha, R) & \alpha \neq \square \\ (q_{Y_2}, \alpha) \mapsto (q_{Y_2}, \alpha, R) & \alpha \neq \square \\ (q_{Y_2}, \square) \mapsto (q_{YE}, \square, L) & \\ (q_{YE}, \alpha) \mapsto (q_{YE}, \square, L) & \alpha \neq \square \\ (q_{YE}, \square) \mapsto (q_Y, \square, L) & \end{array}$$

The last set of commands first moves the head to the blank character immediately following the string on the tape, then erases the string, and enters q_Y. We call the resulting Turing machine *refined*.

3 ETS data structures and additive transforms

We introduce the concepts of a *primitive* representing an elementary constructive step, a *formation* representing a finite constructive process (a finite sequence of elementary constructive steps), and a *struct* representing a single object. As was mentioned in the introduction, objects are thought of as equivalence classes of constructive histories, therefore structs are defined as equivalence classes of formations. The auxiliary concepts of a site, composite, and site replacement are introduced for technical reasons (which are discussed in [8]).

Let S be a countably infinite set whose elements are called *sites*.

Definition 1. A **primitive** is a 3-tuple $\pi \overset{\text{def}}{=} \langle \alpha, I, T \rangle$, where α is the **label of primitive** π, and I, T are disjoint finite linearly ordered subsets of the set of sites S.

For a primitive $\pi = \langle \alpha, I, T \rangle$, we use the following notation:

$$\begin{array}{rcll} \text{init}(\pi) & \overset{\text{def}}{=} & \{i \mid i \in I\} & \text{set of } \textbf{initial sites} \text{ for } \pi \\ \text{term}(\pi) & \overset{\text{def}}{=} & \{i \mid i \in T\} & \text{set of } \textbf{terminal sites} \text{ for } \pi \\ \text{sites}(\pi) & \overset{\text{def}}{=} & \text{init}(\pi) \cup \text{term}(\pi) & \text{set of (all) } \textbf{sites} \text{ for } \pi. \end{array}$$

▶

The reasons for describing elementary constructive steps by the above primitives are discussed in [8, Appendix].

A primitive $\langle \alpha, I, T \rangle$ can be represented pictorially as a convex shape with $|I|$ sites marked on its upper part and $|T|$ sites marked on its lower part. The left-to-right ordering of the sites corresponds the linear orderings on I and T.

Given a Turing machine over alphabet Σ with the set of states Q, we introduce $|Q| + 2|\Sigma| + 3$ primitives shown in Fig. 1 ($q \in Q$, $c \in \Sigma$). The meaning of these primitives, as

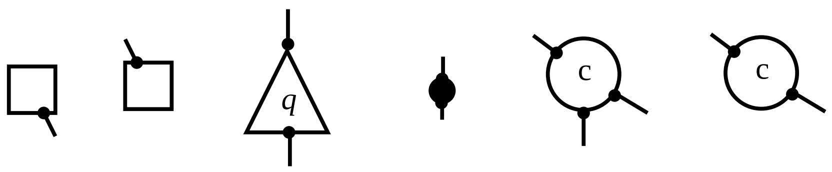

Figure 1: Primitives (site labels are omitted for simplicity).

elementary constructive steps, is the following (starting from the leftmost primitive):

1. Creation of the beginning of a Turing machine tape (formally, the tape is infinite, so this primitive signifies the position before the leftmost non-empty cell of the tape)
2. Creation of the end of a Turing machine tape (the position after the rightmost non-empty cell)
3. ($|Q|$ primitives) Attachment of the head of a Turing machine in state q to the tape ($q \in Q$)
4. Recording that the machine has made a step (this primitive is to be attached at the bottom of the previous one)

5. ($|\Sigma|$ primitives) Creation of an active cell (i.e., the one to which the head points) with a character $c \in C$ inside
6. ($|\Sigma|$ primitives) Creation of an inactive cell with $c \in C$ inside

The initial and terminal sites of each primitive are designed in such a way that the simulation of the Turing machine becomes possible (see below).

Definition 2. A pair of primitives $\langle \pi, \sigma \rangle$ satisfies the **attachment condition**, denoted $\pi \dashv \sigma$, if and only if

$$\text{sites}(\pi) \cap \text{sites}(\sigma) = \text{term}(\pi) \cap \text{init}(\sigma).$$

A **composite** is a finite set of primitives $\gamma = \{\pi_1, \ldots, \pi_n\}$ which can be ordered by a linear ordering $<$ so that for all $i < j$, $\pi_i \dashv \pi_j$.

For the above composite γ, define the following sets of sites:

$$\begin{aligned} \text{init}(\gamma) &= \cup_{i=1}^{n} \text{init}(\pi_i) \setminus \cup_{i=1}^{n} \text{term}(\pi_i) \\ \text{term}(\gamma) &= \cup_{i=1}^{n} \text{term}(\pi_i) \setminus \cup_{i=1}^{n} \text{init}(\pi_i). \end{aligned}$$

▶

Pictorially, it is convenient to represent a composite by connecting the shared sites of its primitives; the attachment condition expresses formally the intuitively necessary requirement that connections can only exist between terminal sites of one primitive and initial sites of another. The difference between composites and hypergraphs is discussed in [8, Section 2.1.2]. In Fig. 2, two examples of composites are shown (site labels are omitted).

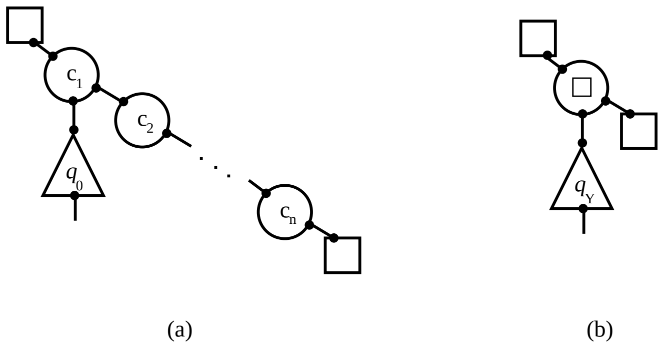

Figure 2: (a) Composite corresponding to the initial configuration of the Turing machine with the input string $c_1 \ldots c_n$; (b) Composite corresponding to the "yes"-configuration of the Turing machine with empty tape.

Definition 3. An ordered pair of composites $\langle \alpha, \beta \rangle$ satisfies the **composition condition**, if for all primitives $\pi \in \alpha$, $\sigma \in \beta$, $\pi \dashv \sigma$.

For a pair of composites $\langle \alpha, \beta \rangle$ satisfying the composition condition, the **composition** $\alpha \lhd \beta$ is defined as the union of sets α and β. ▶

Composition of any pair of composites satisfying the composition condition is a composite. Moreover, the composition operation is associative [8, Lemma 4].

Two composites that differ only in their sites but have the same internal structure represent the same constructive processes. Thus, a constructive process is uniquely represented by the set of all site replacements of a given composite:

Definition 4. For a primitive $\pi = \langle \alpha, I, T \rangle$ and an injective mapping $h : \mathrm{sites}(\pi) \to \mathcal{S}$, called **site replacement**, the primitive $\pi\langle h \rangle$ is defined as

$$\pi\langle h \rangle \overset{\text{def}}{=} \langle \alpha, h(I), h(T) \rangle.$$

For a composite $\gamma = \{\pi_1, \ldots, \pi_n\}$ and an injective mapping $h : \mathrm{sites}(\gamma) \to \mathcal{S}$, the composite $\gamma\langle h \rangle$ is defined as follows:

$$\gamma\langle h \rangle \overset{\text{def}}{=} \{\pi_1\langle h\big|_{\mathrm{sites}(\pi_1)}\rangle, \ldots, \pi_n\langle h\big|_{\mathrm{sites}(\pi_n)}\rangle\}.$$

▶

A **formation** is defined as the set of all composites obtained from a given composite through site replacements. A pictorial representation of a formation can be obtained from that of a composite by removing the site labels. Since formations have no sites, the operations of composition and site replacement are undefined for them. The connection between formations and abstract graphs is discussed in [8, Section 2.1.5].

Next, we define *structs* representing objects via the introduction of the *semantic equivalence relation* on formations. The informal assumptions behind the definitions, as well as the reasons for some formal requirements, are discussed in [8, Section 2.2] and omitted here.

Definition 5. Composite α is called **directly reducible** to composite β via a composite pair $c = \langle \gamma, \gamma' \rangle$, denoted $\alpha \xrightarrow{c} \beta$, if there exist composites γ_1, γ_2 such that

$$\begin{aligned} \alpha &= \gamma_1 \lhd \gamma \lhd \gamma_2 \\ \beta &= \gamma_1 \lhd \gamma' \lhd \gamma_2. \end{aligned}$$

Formation $\bar{\alpha}$ is called directly reducible to $\bar{\beta}$ via c, if there exist composites $\alpha \in \bar{\alpha}$, $\beta \in \bar{\beta}$ such that $\alpha \xrightarrow{c} \beta$.

The direct reduction relation induced by a set of composite pairs $\mathcal{I}$ is defined as follows:

$$\bar{\alpha} \xrightarrow{\mathcal{I}} \bar{\beta} \quad \Longleftrightarrow \quad \exists\, c \in \mathcal{I} \;\; \bar{\alpha} \xrightarrow{c} \bar{\beta}.$$

The **equivalence relation** induced by $\mathcal{I}$, denoted $\sim_{\mathcal{I}}$, is defined as the reflexive, symmetric, and transitive closure of the direct reduction relation $\xrightarrow{\mathcal{I}}$, i.e., the minimal equivalence relation containing $\xrightarrow{\mathcal{I}}$. ▶

Next, we formulate a condition on composite pairs in $\mathcal{I}$, sufficient to make the equivalence relation $\sim_{\mathcal{I}}$ a congruence with respect to composition ([8, Lemma 18]).

Definition 6. A composite pair $c = \langle \gamma, \gamma' \rangle$ is called a **semantic identity**, if

$$\mathrm{init}(\gamma) = \mathrm{init}(\gamma'), \qquad \mathrm{term}(\gamma) = \mathrm{term}(\gamma').$$

We will use the following "equality" notation for the semantic identities: identity $\langle \gamma, \gamma' \rangle$ will be denoted as $\gamma = \gamma'$. ▶

Definition 7. If $\mathcal{I}$ is a set of semantic identities, relation $\sim_{\mathcal{I}}$ is called the **semantic equivalence** relation induced by $\mathcal{I}$. ▶

Definition 8. Let $\mathcal{I}$ be a set of semantic identities, and let $\sim_{\mathcal{I}}$ be the semantic equivalence relation on formations induced by $\mathcal{I}$. For a formation $\bar{\gamma}$, the **struct** containing $\bar{\gamma}$ is defined as

$$\boldsymbol{\gamma} \overset{\text{def}}{=} [\bar{\gamma}]_{\sim_{\mathcal{I}}} \overset{\text{def}}{=} [\![\gamma]\!]_{\sim_{\mathcal{I}}} \overset{\text{def}}{=} \{\bar{\gamma}' \mid \bar{\gamma}' \sim_{\mathcal{I}} \bar{\gamma}\}.$$

▶

We impose two requirements on the set $\mathcal{I}$ of semantic identities: first, that $\mathcal{I}$ is finite, and second, that every equivalence class w.r.t. $\sim_{\mathcal{I}}$ (i.e., every struct) is finite. The latter condition ensures that all operations in the ETS formalism (including checking struct equality) are computable [8, Section 2.3.7]—which is a necessary requirement for the purpose of this paper: the simulation of Turing machines by ETS operations. Other reasons for imposing this requirement of *struct finiteness* are discussed in [8].

Given a Turing machine M with alphabet Σ, set of states Q, and transition mapping

$$\delta : \Sigma \times (Q \setminus \{q_Y, q_N\}) \to \Sigma \times Q \times \{L, R\},$$

we introduce the semantic identities shown in Fig. 3. They are designed in such a way that all formations representing various configurations that the machine can enter starting from a fixed initial configuration are semantically equivalent.

The detailed meaning of the semantic identities in Fig. 3 is the following.

- (R1) This is a family of $|\Sigma| \cdot (|\Sigma| - 1) \cdot (|Q| - 2) \cdot (|\Sigma| - 1) \cdot (|\Sigma| - 1) \cdot |Q|$ identities, each of which signifies that composite representing the configuration in which
  - the head points to a character $c \in \Sigma$
  - the machine is in state $q \in Q \setminus \{q_Y, q_N\}$
  - there is at least one character $\alpha \in \Sigma \setminus \{\square\}$ to the right of c

  is equivalent to the composite representing the configuration in which
  - c is replaced by a character $c' \in \Sigma \setminus \{\square\}$
  - the machine is in state $q' \in Q$
  - the head points to α
  - the fact that the machine has made the step $(q, c) \mapsto (q', c', R)$ is recorded via the attachment of the step recording primitive at the bottom to the head primitive.
- (L1) This family of identities is similar to R1 and corresponds to the step $(q, c) \mapsto (q', c', L)$.
- (R2) Composite representing the configuration in which
  - the head points to a character $c \in \Sigma$
  - the machine is in state $q \in Q \setminus \{q_Y, q_N\}$
  - c is the last character on the tape

  is equivalent to the composite representing the configuration in which
  - c is replaced by a character $c' \in \Sigma \setminus \{\square\}$
  - the machine is in state $q' \in Q$

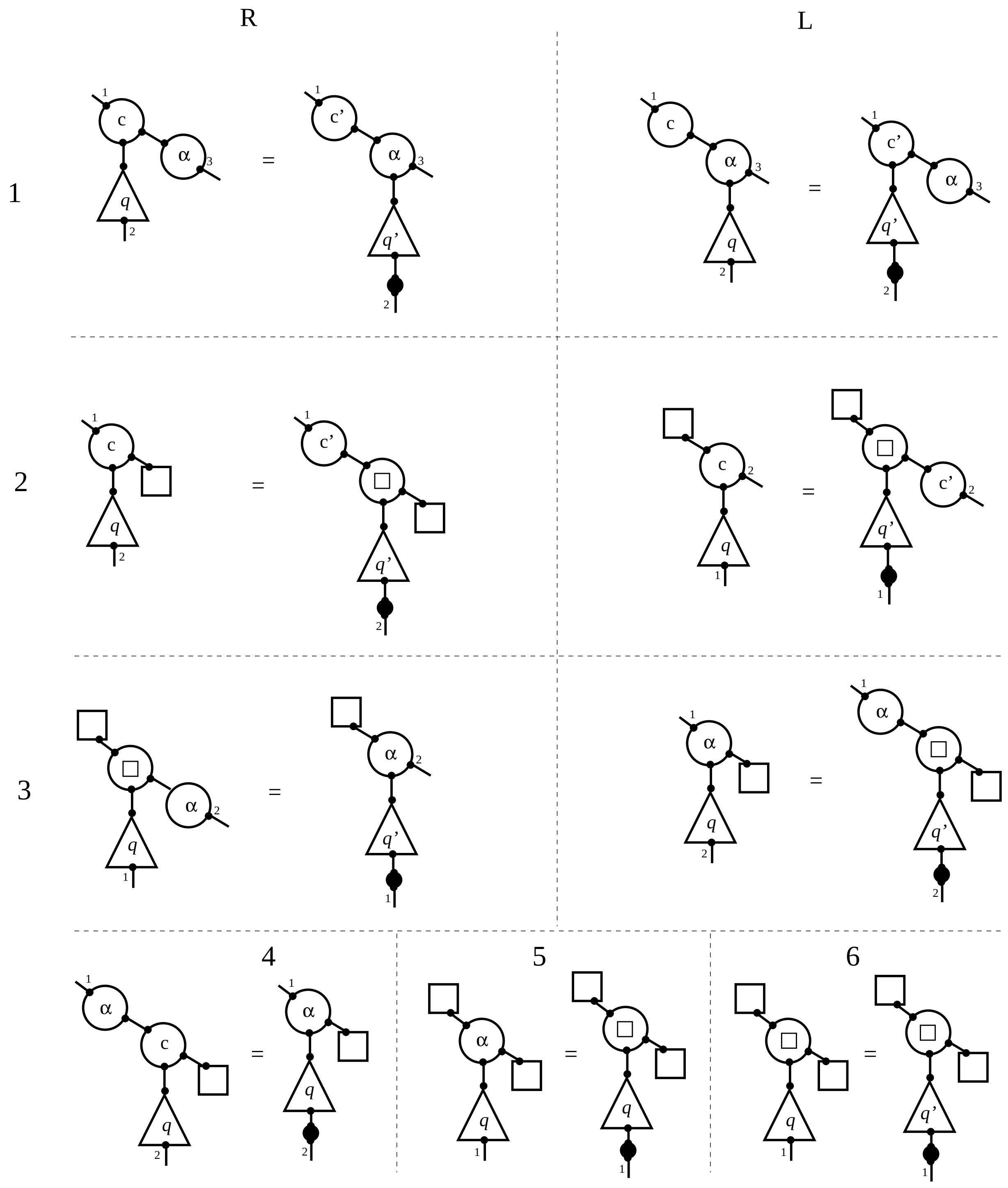

Figure 3: Semantic identities.

  - a new cell is created after c with a $\Box$ symbol in it
  - the head points to the new cell
  - the fact that the machine has made the step $(q, c) \mapsto (q', c', R)$ is recorded.

- (L2) is similar to (R2).
- (R3) Composite representing the configuration in which the head points to the $\Box$ symbol at the beginning of the tape, the machine is in state q, and there is a character $\alpha \in \Sigma$ to the right of the head is equivalent to the one in which the cell with the $\Box$ symbol is removed, the machine is in state q', the head points to α, and the step $(q, \Box) \mapsto (q', \Box, R)$ is recorded.
- (L3) is similar to (R3).

Identities (4–6) simulate the final erasing stage of a refined Turing machine:

- (4) Composite representing the configuration in which the head points to the rightmost symbol $c \in \Sigma \setminus \{\Box\}$, the machine is in state $q \in \{q_{YE}, q_{NE}\}$, and there is a character α to the left of c is equivalent to the one in which the cell with c is removed, the machine is in state q again, the head points to α, and the step $(q, c) \mapsto (q, \Box, L)$ is recorded.
- (5) Composite representing the configuration in which the head points to the only symbol $\alpha \in \Sigma \setminus \{\Box\}$ on the tape and the machine is in state $q \in \{q_{YE}, q_{NE}\}$ is equivalent to the one in which α is replaced by $\Box$, the machine is in state q again, and the fact that the machine has made the step $(q, \alpha) \mapsto (q, \Box, L)$ is recorded.
- (6) Composite representing the configuration in which the tape is empty and the machine is in state $q \in \{q_{YE}, q_{NE}\}$ is equivalent to the one in which the tape is empty again, the machine is in the corresponding state $q' \in \{q_Y, q_N\}$, and the step $(q, \Box) \mapsto (q', \Box, L)$ is recorded.

The direct reduction relation and the semantic equivalence relation induced by the identities in Figs. 3 are denoted $\to_M$ and $\sim_M$, respectively.

Next, we define additive context-dependent transforms on composites, formations, and structs.

Definition 9. A pair of composites $\tau = \langle \alpha, \beta \rangle$ satisfying the composition condition is called a **transform** with the **context** α and **body** β, denoted

$$\text{context}(\tau) \overset{\text{def}}{=} \alpha, \qquad \text{body}(\tau) \overset{\text{def}}{=} \beta.$$

If α is empty, the transform is called **context-free**. ▶

Definition 10.

1. A transform τ is **applicable to a composite** γ, if the following two requirements are met:
   - there exist composites γ_1, γ_2 such that $\gamma = \gamma_1 \lhd \text{context}(\tau) \lhd \gamma_2$
   - composites γ and body(τ) satisfy the composition condition.

2. Transform τ is **applicable to a formation** $\bar{\gamma}$, if τ is applicable to a composite $\gamma \in \bar{\gamma}$.
3. Transform τ is **applicable to a struct** $\boldsymbol{\gamma}$, if τ is applicable to a formation $\bar{\gamma} \in \boldsymbol{\gamma}$.

▶

The $|Q| + 1$ transforms corresponding to the Turing machine are shown in Fig. 4.

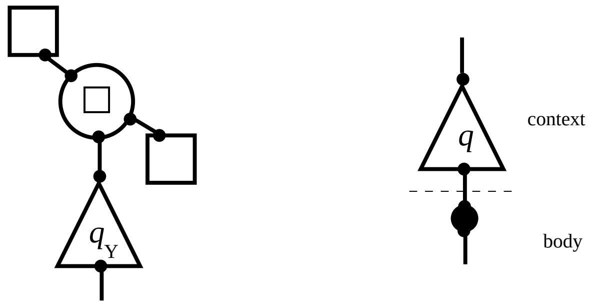

Figure 4: Transforms corresponding to the Turing machine (the first one is context-free).

Definition 11. A finite sequence of transforms $\langle \tau_1, \ldots, \tau_n \rangle$ is called **valid**, if

- composition $\gamma = \text{body}(\tau_1) \lhd \ldots \lhd \text{body}(\tau_n)$ exists
- τ_1 is context-free
- for all $i \in \{2, \ldots, n\}$, τ_i is applicable to struct

$$[\![\text{body}(\tau_1) \lhd \ldots \lhd \text{body}(\tau_{i-1})]\!].$$

The **result** of the above valid sequence of transforms is defined as struct $[\![\gamma]\!]$. ▶

Classes of structs are specified by transforms as follows.

Definition 12. For a transform τ, an injective mapping

$$f : \text{sites}(\text{context}(\tau) \lhd \text{body}(\tau)) \to S$$

is called a **transform site replacement**. By definition,

$$\tau\langle f \rangle = \langle \text{context}(\tau)\langle f \rangle, \text{body}(\tau)\langle f \rangle \rangle.$$

An **abstract transform** $\bar{\tau}$ is defined as the set of all transforms obtained from τ by site replacements:

$$\bar{\tau} = \{\tau\langle f \rangle\}.$$

▶

Definition 13. Let $\bar{T}$ be a finite set of abstract transforms. The **class** of structs generated by $\bar{T}$, $\langle \bar{T} \rangle$, is defined as the set of results of all valid sequences of transforms $\langle \tau_1, \ldots, \tau_n \rangle$, where $\bar{\tau}_i \in \bar{T}$. ▶

Denote by C_M the class of structs generated by the abstract transforms $\bar{\tau}_Y$ and $\bar{\tau}_q$ ($q \in Q$) shown in Fig. 4. If the struct finiteness condition is satisfied, there exists an algorithm that checks whether a struct $\boldsymbol{\gamma}$ belongs to a given class $\langle \bar{T} \rangle$. Indeed, for all formations $\bar{\gamma} \in \boldsymbol{\gamma}$, one can take any composite $\gamma \in \bar{\gamma}$ and consider the set of all decompositions

$$\gamma = \gamma_1 \lhd \ldots \lhd \gamma_k,$$

where each γ_i coincides with the body of a transform τ ($\bar{\tau} \in \bar{T}$). For each decomposition, one can verify whether the corresponding sequence of transforms is valid. Then $\boldsymbol{\gamma} \in \langle \bar{T} \rangle$ if and only if at least one such sequence can be found. Thus, classes of structs are recursive. We will prove the converse statement, which implies that all recursive languages can be represented as classes of structs.

Theorem 1. *Let $L \subset (\Sigma \setminus \{\square\})^*$ be a recursive language, and let M be a refined Turing machine accepting it. Let $s \in (\Sigma \setminus \{\square\})^*$ be a string encoded by the composite γ_s as shown in Fig. 2(a). Then*

$$s \in L \iff [\![\gamma_s]\!] \in C_M.$$

Proof. According to the set of semantic identities introduced in Figs. 3, the formations shown in Fig. 5 are semantically equivalent. This chain of formations corresponds to the

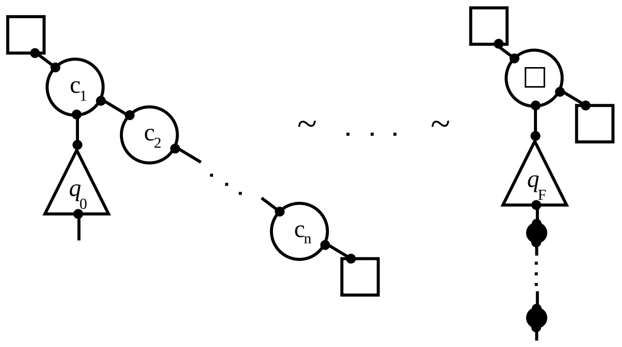

Figure 5: Chain of equivalent formations corresponding to a computation of the Turing machine.

computation of the refined Turing machine M, starting from the initial configuration with the input string s and halting in state $q_F \in \{q_Y, q_N\}$, where $q_F = q_Y$ if and only if $s \in L$. Since M halts after finitely many (say, m) steps, the above chain is finite, and the last formation $\bar{\gamma}_F$ contains m step recording primitives. Clearly, the struct $[\bar{\gamma}_F] = [\![\gamma_s]\!]$ belongs to the class C_M if and only if $q_F = q_Y$. ■

4 Proof of struct finiteness

In order to show that the above representation of Turing machines by transforms implies their Turing-completeness, we need to prove that the semantic equivalence relation $\sim_M$ satisfies the struct finiteness condition. In fact, we only need to prove that structs containing a string formation shown in Fig. 2(a) (we call them *string structs*) are finite. We do it in two steps:

1. Prove that every string struct contains a canonical irreducible formation.
2. Show that the existence of this canonical formation implies struct finiteness.

Let A be the union of all string structs. Note that every formation in a string struct represents a configuration of the Turing machine. Indeed,

- the string formation shown in Fig. 2(a) represents the initial configuration
- application of any semantic identity in Fig. 3 from left to right corresponds to a step of the Turing machine
- application of any semantic identity from right to left corresponds to a backward step of the Turing machine.

Since, for any configuration, the step of the Turing machine is uniquely determined by the value of the transition mapping, every formation in a string struct can be directly reduced to at most one formation. In particular, this trivially implies that the direct reduction relation on A is *confluent*:

Definition 14. Let $\to^*$ denote the reflexive and transitive closure of relation $\to$. Relation $\to$ is called *confluent* [2, Def.1.1.6], if whenever $w \to^* x$ and $w \to^* y$, there exists z such that $x \to^* z$ and $y \to^* z$. ▶

According to [2, Corollary 1.1.8], confluence implies that every string struct contains at most one irreducible formation. It also contains one of the formations corresponding to the "yes"- or "no"-configuration of the Turing machine (see Fig. 2(b)), which are irreducible. Call these formations *canonical*. Due to confluence, every other formation in the string struct is reducible to the canonical one. Therefore, it is sufficient to show that only finitely many formations can be reducible to it. Moreover, for every formation there exist only finitely many formations directly reducible to it, since for every configuration of the Turing machine there are finitely many backward steps. Hence, it is sufficient to shown that there is no infinite chain of backward direct reductions starting from the canonical formation:

$$\bar{\gamma}_0 \leftarrow_M \bar{\gamma}_1 \leftarrow_M \bar{\gamma}_2 \leftarrow_M \cdots$$

But the latter follows from the fact that, according to the semantic identities in Fig. 3, whenever $\bar{\alpha} \to_M \bar{\beta}$, the number of step recording primitives (the fourth one in Fig. 1) in $\bar{\alpha}$ is one less than in $\bar{\beta}$.

5 Conclusion

The statement proved in this paper, that ETS additive transforms are Turing-complete, means that the generative power of various equivalent computational formalisms (such as the Turing machine) is equal to that of the ETS formalism. In other words, all classes of real-world objects that we attempt to describe in algorithmic terms (for example, as string languages generated by a Chomsky grammar), can also be described in terms of ETS transforms.

However, we do not suggest that, if one wants to represent a class of real-world objects, one should first design an algorithmic description of this class, and then translate it into the ETS language by applying the construction described in this paper. In our opinion, such an approach is not feasible because computational formalisms were not designed for the purpose of object representation and class description. In contrast, the ETS formalism offers a mechanism for constructing representations of complex objects and describing their classes. This mechanism is based on a multi-level representational hierarchy and the "intelligent process" that constructs this hierarchy and subsumes the process of inductive learning [6]. Thus, our research direction is to study this process and apply it to particular representation and classification problems. The result presented in this paper guarantees that in doing so we are not losing any computational power.

References

[1] J.M. Abela, ETS Learning of Kernel Languages, Ph.D. Thesis, Faculty of Computer Science, University of New Brunswick, 2002.

[2] R. Book, F. Otto, *String-Rewriting Systems*, Springer-Verlag, New York, Berlin, 1993.

[3] N. Chomsky, *Syntactic Structures*, Fifth printing, Mouton & Co., London, 1965.

[4] L. Goldfarb, Is there a different mathematics, "mathematics of the mind", that would explain the biological (structural) "measurement" processes? Lev Goldfarb's home page, `http://www.cs.unb.ca/profs/goldfarb`

[5] L. Goldfarb, On the foundations of intelligent processes I: An evolving model for pattern learning, *Pattern Recognition 23*, 595–616, 1990.

[6] L. Goldfarb, D. Gay, O. Golubitsky, D. Korkin, What is a structural representation? Second version. Faculty of Computer Science, U.N.B., Technical Report TR04-165, 2004.

[7] L. Goldfarb, O. Golubitsky, D. Korkin, What is a structural representation? Faculty of Computer Science, U.N.B., Technical Report TR01-137, 2001.

[8] O. Golubitsky, On the Formalization of the Evolving Transformation System Model, Ph.D. Thesis, Faculty of Computer Science, University of New Brunswick, 2004.

[9] *Handbook of Formal Languages*, eds. G. Rozenberg, A. Salomaa, Springer-Verlag, Berlin, Heidelberg, 1997.

[10] E. Weisstein, Turing Machine, From MathWorld–A Wolfram Web Resource, `http://mathworld.wolfram.com/TuringMachine.html`